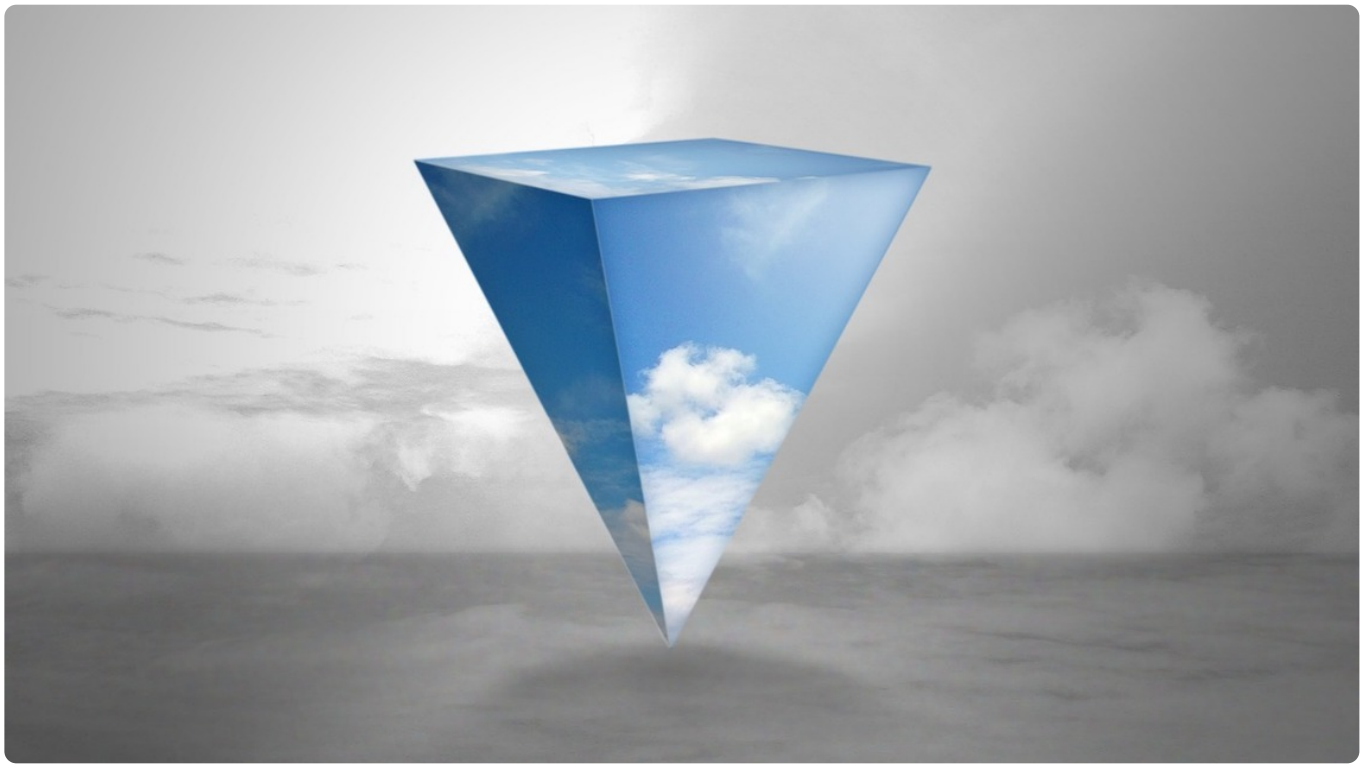
19 | 链式 & 比较 & 环式学习法：怎么多维度提升技术能力？

2021-01-11 李运华

大厂晋升指南 进入课程 >



讲述：安晓辉
时长 16:35 大小 15.19M

你好，我是华仔。

上一讲我介绍了三段分解法，教你把“10 年成为大牛”这个宏大的目标，分解成 1～2 个月的可落地计划，然后再按周来执行实际的行动。

但是，不同级别的**核心要求**是不一样的，晋升时评委的**考察重点**也不一样，所以在成长过程中，我们主要提升的**技术维度**也在发生变化。

一般来说，P5/P6/P7 主要提升**技术深度**，P7/P8 主要提升**技术宽度**，P8/P9 主要提升**技术广度**。（我在第 11 讲和第 16 讲分别以前端和 Java 后端为例，解释了这三个维度的区别。）

这三个不同的技术维度，分别适合用不同的方法来提升，这一讲我就会为你一一介绍，让你的学习更有针对性，在回答晋升评委提问的时候也能做到游刃有余。

链式学习法：提升技术深度

提升技术深度，最好使用**链式学习法**。

如果你参加过晋升答辩，一定经历过评委的“追命连环问”，比如：

1. 你在讲解 PPT 的时候提到，某个项目使用了 Netty 技术，评委首先会问你 Netty 的一些技术点；
2. 当你回答说 Netty 的本质是 Reactor 网络模型时，评委又会问你 Reactor 网络模型的原理；
3. 当你回答说 Reactor 的基础是 Java NIO 的时候，评委又会问你 Java 的 NIO/BIO 的技术细节；
4. 当你回答说 Java 的 NIO 在 Linux 平台上是基于 epoll 来实现时，评委又问你 Linux 的 epoll/select 等的原理。

……

面对这种“打破砂锅问到底”的方式，如果平时没有充足的准备，你很可能会卡住。

所谓“链式学习法”，顾名思义，就是学习的过程好像从水里拉起一根链条，拉出一环后面又接着一环，最后将整个链条全部拉出来。

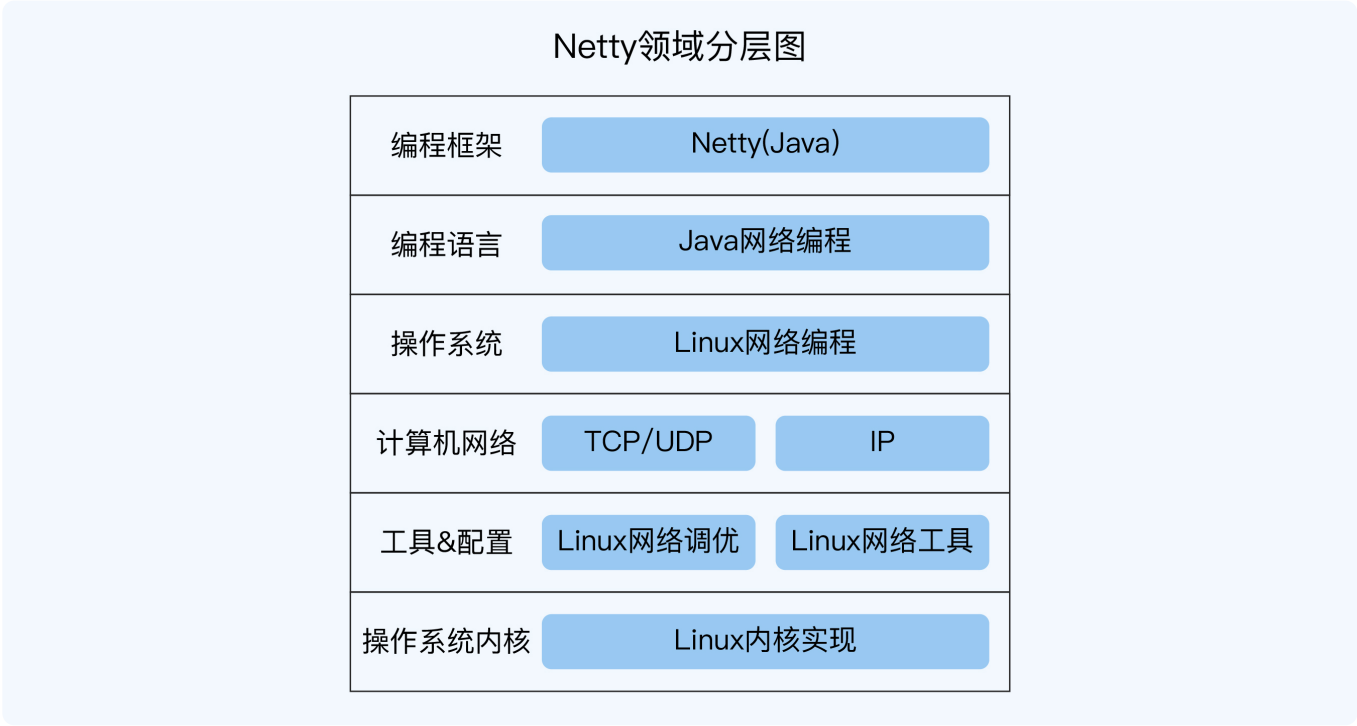
当知识联结成锁链，**环环相扣**，你对技术的理解就很透彻，评委问到底，你就能答到底。

但是知识的锁链不是胡乱连接的，环环相扣的方式很有讲究。常见的方式有两种：

第一种是**自顶向下、层层关联**，打通一项技术的领域分层。

第二种是**由表及里、层层深入**，打通一项技术的细节分层。

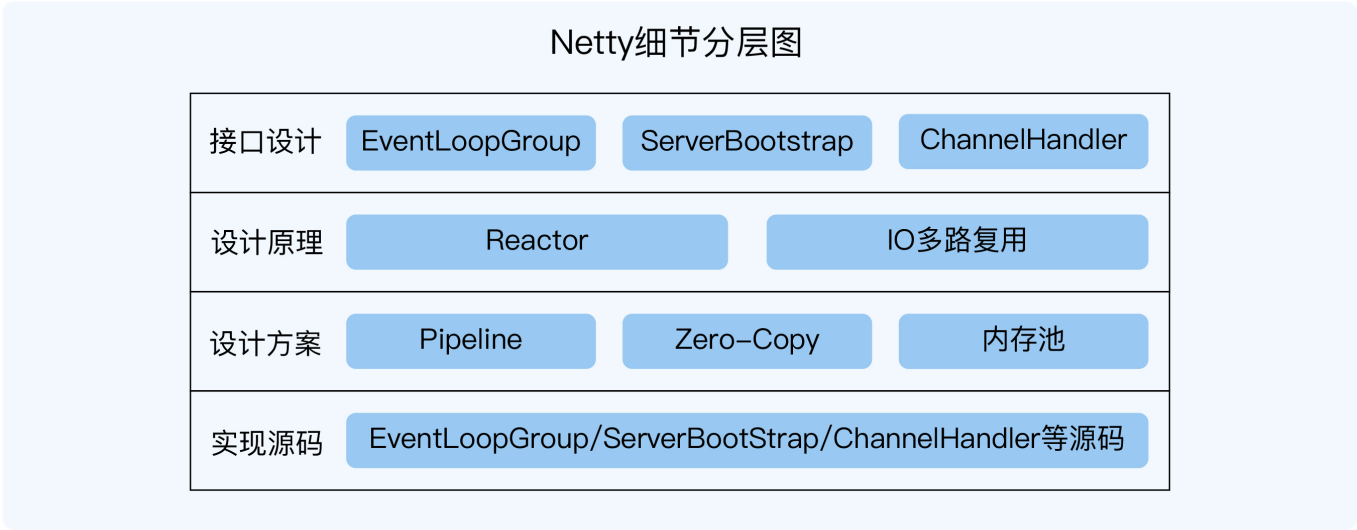
以 Netty 网络编程为例，相关领域一共可以分为 6 层，要么上层依赖下层，比如 Netty 依赖 Java 网络编程，Java 网络编程在 Linux 上又依赖 Linux 提供的网络编程接口；要么下层是上层的应用和实现，比如 TCP/IP 是原理，而 Linux 网络调优和工具是 TCP/IP 的具体应用。它的领域分层图如下所示：



注：

1. 以上图示仅为示例，你需要根据自己的实际工作情况来分析。如果你是在 Windows 平台上做开发，那么上图“操作系统”这一层就要改为“Windows 网络编程”，“工具 & 配置”这一层就要改为“Wireshark”之类的。
2. 具体分层关系并没有业界统一的标准，比如“工具 & 配置”这一层，如果你认为应该放在“计算机网络”那一层的上面，其实也是可以的。

同样以 Netty 网络编程为例，技术细节可以分为 4 层，它的细节分层图如下所示：



链式学习法的步骤

链式学习法的第一步，就是要**明确一项技术的深度可以分为哪些层**。

具体来说，就是画出“领域分层图”和“细节分层图”。一开始你可能会觉得画不出来，这恰恰说明你对深度的理解还不够，而尝试画图本身就是一个梳理结构、强化认知的过程。

画出了两张图之后，第二步就是要**明确你自己要学到哪一层**。

学得太浅，达不到提升深度的目的；学得太深，又会耗费太多的时间和精力。以 Netty 网络编程为例，从我自己实践和指导别人的经验来看，领域分层图的 6 层不用都学，大部分人学个 3～5 层就够了；不过细节分层图的 4 层，还是建议你每一层都学。

确定学到哪一层之后，第三步就是要**明确每一层应该怎么学**。

在领域分层图中，越往上越偏应用，实际工作中用得越多，越往下越偏原理（包括相关的工具和配置），实际工作中用得越少。所以总的原则是，在上层投入更多时间，更关注细节和熟练使用，在下层投入相对少的时间，更加关注原理和简单应用。

比如对于 Netty 网络的领域分层图，如果你不是 Netty 项目的开发人员，而是只想使用 Netty 来搭建自己的系统，那么“Linux 网络编程”这一层，你只要掌握 select/epoll 等技术原理和优缺点就行了，epoll 提供的 API 你有时间可以大概看看，没时间不看也可以；但是对于 Netty 本身提供的 API，则是越熟练越好。

在细节分层图中，你需要详细地学习每一层。要注意的是，对于“实现源码”这一层，你不需要去掌握每一行源码，只要掌握关键源码就行了，也就是和设计原理以及设计方案相关的源码。

链式学习法的优点

链式学习法主要有两个优点：

1. 促使我们主动提升

大部分人在实际工作中，很多技术都只接触到了领域分层图和细节分层图中的前 2 层，没有进一步地去了解。

而如果采用链式学习法，你就会意识到，使用一项技术完成了工作，并不意味着你就完全掌握了这项技术。你还需要把刚刚自己用到的技术作为切入点，画出完整的领域分层图和细节分层图，然后逐一攻破，这样才能提升深度，达到精通水平。

2. 将知识和技能系统化

明确知识和技能点之间的关联关系，有助于更好的理解和应用这些知识和技能。

例如，如果我们要在 Linux 平台上基于 Netty 开发并发 10 万连接的高性能服务器，既要深入掌握 Netty 的技术细节，又要深度掌握领域深度相关的技术，包括：

Netty 技术细节：需要设置 Netty 的相关参数（ChannelOption.SO_BACKLOG，ChannelOption.TCP_NODELAY，ChannelOption.SO_REUSEADDR 等）。

Java 网络编程：调试的时候需要知道 Java 的网络编程 API 等等。

Linux 网络工具：需要使用 Linux 网络工具定位问题。

Linux 操作系统配置：需要修改 Linux 的最大文件句柄数、需要优化 Linux 的 TCP/IP 参数（net.ipv4.tcp_tw_reuse，net.ipv4.tcp_keepalive_time 等）。

只有使用链式学习法，你才能系统地了解到这些关联的知识和技能，以及如何将它们串起来。

链式学习法小结

现在，我们回顾一下链式学习法的重点：

1. 链式学习法是让知识形成锁链，环环相扣，主要用来提升技术深度。
2. 链式学习法的步骤包括：明确一项技术的深度可以分为哪些层，明确要学到哪一层，明确每一层应该怎么学。
3. 链式学习法的优点有：促使我们主动提升，将知识和技能系统化。

比较学习法：提升技术宽度

提升技术宽度，最好使用**比较学习法**。

如果你有过晋升 P7 或者更高级别的经历，肯定被问到过大量跟“Why”有关的问题，比如：

1. 为什么选择 Redis，为什么不用 Memecached?
2. 为什么选择 MySQL 而不是 Redis?
3. 选择 Flink 的理由是什么？（除了 Flink 本身的技术特点外，还需要你回答为什么选择 Flink 而不是 Spark 或者 Storm。）

……

这些问题大部分都是考察你思考、判断和决策的逻辑和过程。如果你只有技术深度而没有技术宽度，这时就会陷入窘境：单个技术细节你都很熟悉，但是却无法解释为什么用这个，而不用那个。

所谓比较学习法，就是**横向比较**同一个领域中类似的技术，梳理它们异同，分析它们各自的优缺点和适用场景。

这样你就能加深对整个领域的理解，评委问的每个为什么，你都能回答得有理有据。

比较学习法的步骤

比较学习法的具体操作步骤如下：

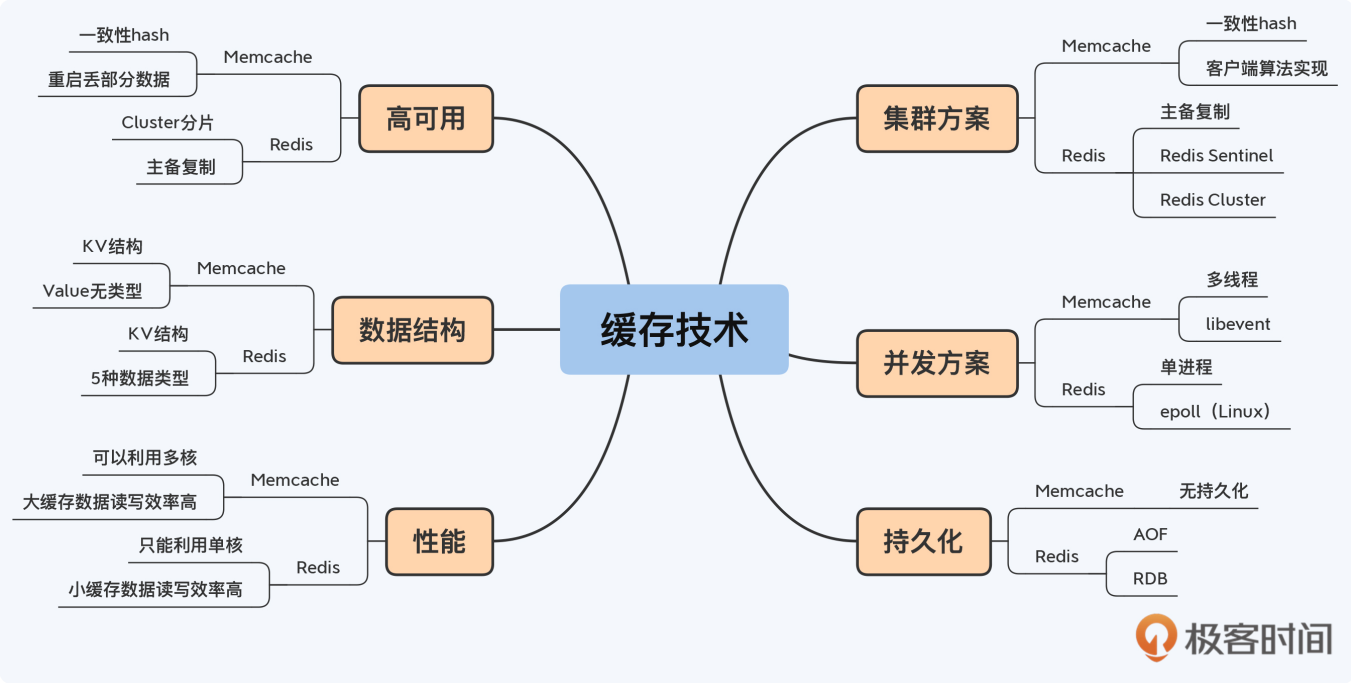
1. 先用链式学习法掌握某个领域的一项技术，将这个领域的关键技术点整理成表格。
2. 基于整理好的技术点，学习这个领域的另一项技术，将它们在技术点上的差异整理成思维导图。
3. 找出差异较大的技术点，将背后的原理和对应用场景的影响整理成表格。

接下来，我以缓存领域的 Memcache 和 Redis 为例，说明一下比较学习法的用法。

1. 先用链式学习法掌握 Memcache 技术，整理出缓存领域的 6 个关键技术点。

技术点	说明
集群方案	是否支持集群，集群的实现原理和方案
并发方案	进程线程模型，网络模型
持久化	是否支持持久化，存储格式，存储方案
高可用	主备，集群，故障恢复
数据结构	数据结构设计
性能	性能测试结果，各种场景性能对比

2. 基于这 6 点快速掌握 Redis 技术，整理出 Memcache 和 Redis 在这些点上的差异。



3. 找出差异较大的技术点，包括并发方案、数据结构、高可用和持久化，整理出它们背后的原理和对应用场景的影响。

差异点	原理	应用场景
并发方案	Redis用单进程可以做无锁设计；Memcache需要用CAS锁，更复杂一些。 但是Redis会出现某个大缓存对象的读写影响其它缓存的操作。	如果有较多大的缓存对象读写，就用Memcache； 否则，用Redis。
数据结构	Redis定位就是结构化的KV存储系统；Memcache只是缓存系统	如果是数据库+缓存这种组合方案就能满足需求，用Memcache更简单一些； 如果业务上有MySQL难以满足而用Redis可以很好满足的强需求，用Redis。
高可用/持久化	Redis定位就是结构化的KV存储系统；Memcache只是缓存系统。	如果缓存生成的代价很高，丢失后可能引起严重的系统问题，用Redis； 如果短时间丢失部分缓存影响不大，则用Memcache。

注：表格内容仅为示例，实际内容不止这么多，如果你有兴趣，可以上网搜索或者自行补充完整。

比较学习法的优点

比较学习法主要有三个优点：

1. 学得快

同一个领域的技术在功能上大都是类似的，区别往往在于实现方案和细节。所以当你掌握了一项技术之后，再去同一个领域的另一项技术，就不需要从 0 开始了，因为基础的部分你已经学会了，只要重点关注它们的差异点就能够快速掌握。

2. 学得全

整理关键技术点和制作思维导图的过程，会促使你把一个领域的技术体系化，更全面、更系统地掌握这个领域。

3. 学得深

从差异点到背后的原理再到应用场景的思考过程，会让你对技术的取舍之道理解得更深，在每一次技术选择时都能给出让人信服的理由。

比较学习法小结

现在，我们回顾一下比较学习法的重点：

1. 比较学习法是横向对比，让选择有理有据，主要用来提升技术宽度。
2. 比较学习法的步骤包括：整理领域关键技术点，整理不同技术的差异点，整理差异点背后的原理和对应用场景的影响。
3. 比较学习法的优点有：学得快，学得全，学得深。

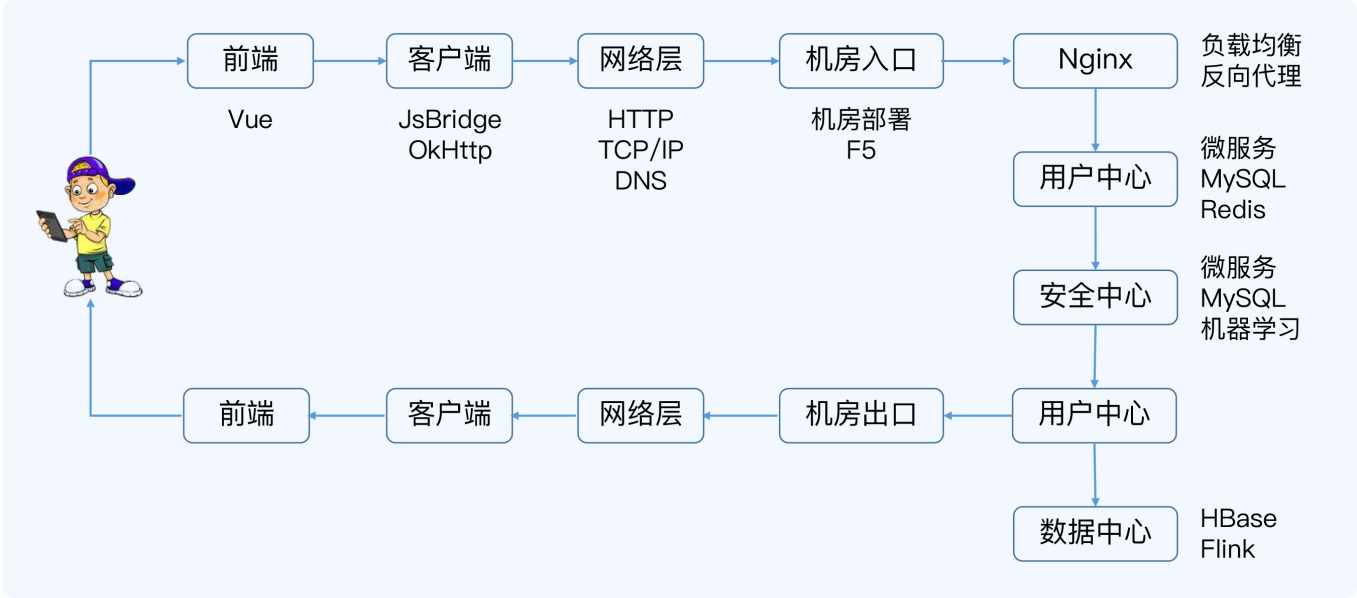
环式学习法：提升技术广度

提升技术广度，最好使用环式学习法。

很多人一听要提升广度，就以为学得越多越好，想到什么牛就学什么，看到什么热就追什么。学了一段时间，感觉学了很多，但好像啥也不会，网撒得很广，却没捞到几条鱼。

所谓环式学习法，就是构建一个完整的闭环过程，将多个领域的“鱼”一网打尽。

技术上常见的闭环是**功能环**，代表某个功能的处理过程。以一个最简单的“用户登录”为例，如果它的实现方式是前端在手机 App 上用做登录页面，后端用了微服务架构来存储，那么就可以构建这样一个功能环：



注意：

1. 上图仅为示意，你可以根据实际情况自己完善，比如拆分为更多环，或者每个环增加更多的技术点。
2. 上图我是用 PPT 画的，你也可以根据自己的喜好采用其他画图工具，比如 UML 类工具。

这里要说明一点，环式学习法更加适合**业务系统相关**的技术人员，而不太适合**中间件**（数据库、缓存、消息队列和服务中心等）相关的的技术人员，因为中间件的技术更加专注于深度和宽度，和具体的业务关系不大，对技术广度的要求并不高。

当然，如果你已经达到了 P8+/P9 这个级别，无论什么领域，都可以采用环式学习法来学习跨领域的技术。

除了功能环以外，还有很多构建闭环的思路，比如业务上常见的“业务环”，它代表某个业务的处理步骤，以及管理上常见的“流程环”，它代表某件事情的处理步骤。

所以，环式学习法不但可以用来提升技术广度，也可以用来提升业务能力和管理水平。

环式学习法的步骤

环式学习法的第一步，就是**把闭环画出来**。

具体的画法是将完整的闭环分为几个关键的环节，然后标出每个环节的关键内容。

就拿“用户登录”这个功能环来说，它可以分为前端、客户端、网络层、机房入口、Nginx、用户中心、安全中心和数据中心，总共 8 个环节；每个环节又会涉及不同的技术，比如客户端涉及 JsBridge 和 OkHttp，用户中心涉及微服务、MySQL 和 Redis 等，总共涉及的技术有 18 项。

通过这么一个简单的功能环，你就可以看出技术广度的边界和范围；而且这些技术都是业务上实际用到的，你完全不用担心自己是没有目的地乱学。

环式学习法第二步，就是**由近及远，逐步攻克闭环上的各个节点**。

就算是同一个闭环，不同领域的人学习顺序也是不同的。还是以用户登陆这个功能环为例，前端的人先需要学客户端的 JsBridge 和 OkHttp 等知识，然后再去学服务端相关的知识；而服务端用户中心的人，需要先学 Nginx 和安全中心相关的知识，之后再逐步扩展到客户端和前端。

通常来说，职业等级越高，技术广度的要求也越高，所以功能环上要求掌握的相关技术也越多。

对于单个技术，你还是需要用链式学习法来学习，但是因为数量太多，全部严格按照链式学习法的要求来学是不太现实的。我的建议是，可以先不去研究源码，只要学习接口设计、设计原理、设计方案这 3 层就行了；在合适的时候或者有时间的时候，可以看看核心源码加深理解。

提升业务能力也很重要

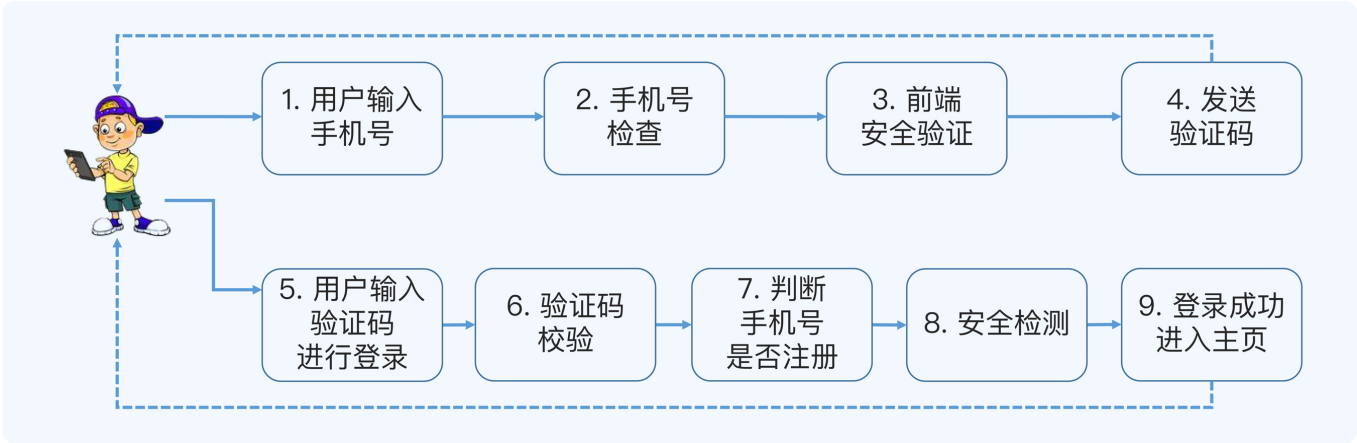
很多技术人员有一个误区，认为业务设计是产品经理的事情，产品经理设计好了，技术人员再把自己负责那部分做好就行了。

这种想法会让你在工作中非常被动，而且可能吃大亏。常见的吃亏场景包括：

1. 讨论需求的时候，因为不懂业务，就算产品的业务需求不合理、实现代价很高，你也发现不了。结果到了设计甚至是编码阶段，你才发现自己做得累死累活，效果还不好。
2. 处理线上故障的时候，因为不熟悉业务，只能被动接受别人的分析和推断，很容易背锅。
3. 因为不熟悉业务，无法承担整体需求分析和方案设计这种任务，导致个人能力得不到锻炼，失去很多晋升机会。

无论是前端、客户端还是服务端的技术人员，最好都花点时间，通过业务环来了解业务的整个流程。

以下是用户登录的业务环，供你参考。



注意：上图仅为示例，省略了很多分支和细节，实际的业务流程图比这个要复杂，你可以直接参考产品经理的需求文档。

环式学习法的优点

环式学习法有两个优点：

1. 培养全局视野

在画出完整闭环的过程中，你可以端到端地了解全流程涉及哪些系统或者模块，每个模块的关键技术是什么，从而培养出全局的视野和能力。

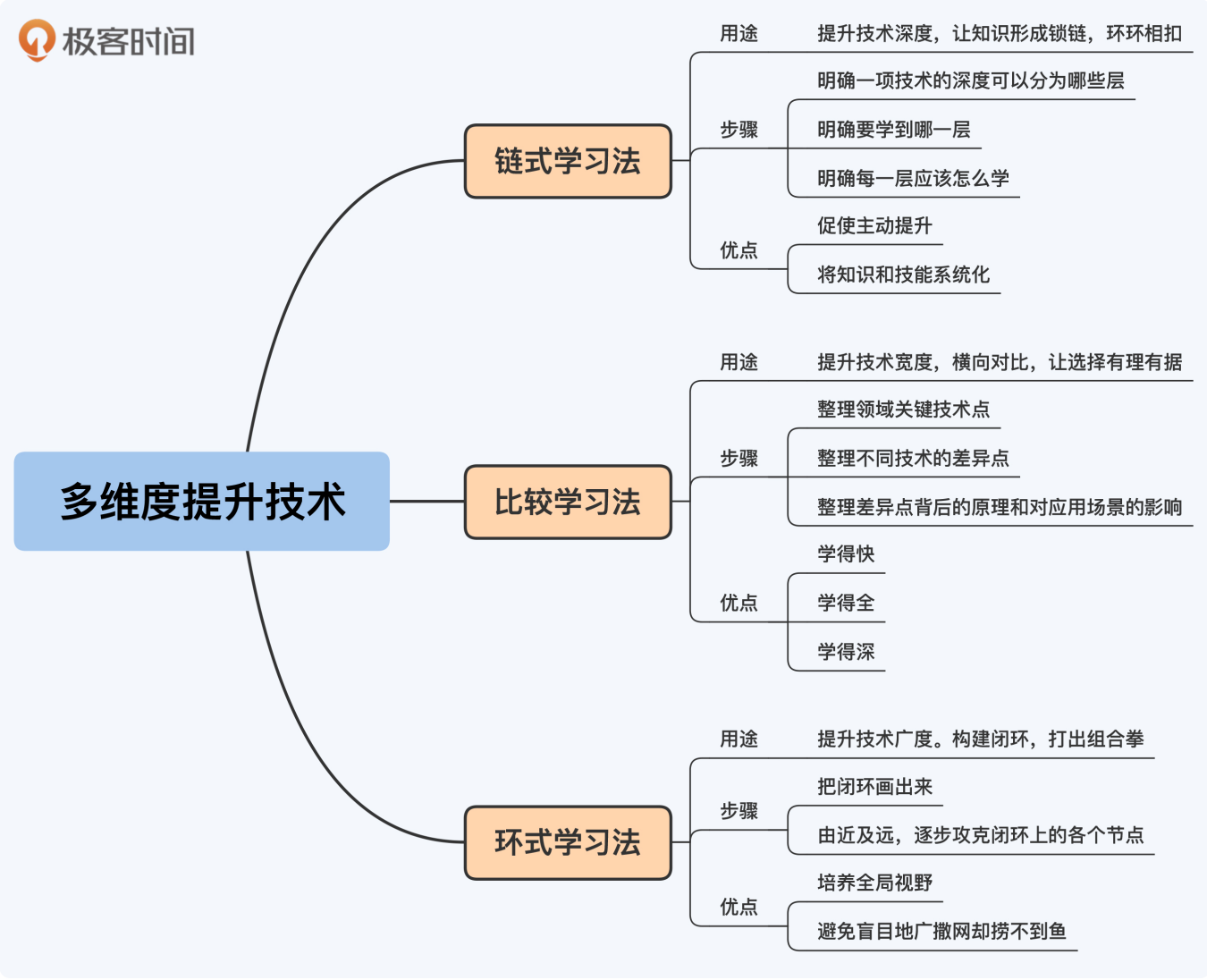
2. 避免盲目地广撒网却捞不到鱼

环式学习法划定的范围是实际工作的闭环，能够形成一套有效的组合拳，而不是东一榔头西一棒槌的胡乱搭配，能够大大提升学习效率。所以你只要对照环来提升就可以了，不用再担心广撒网却捞不到鱼了。

环式学习法小结

现在，我们回顾一下环式学习法的重点：

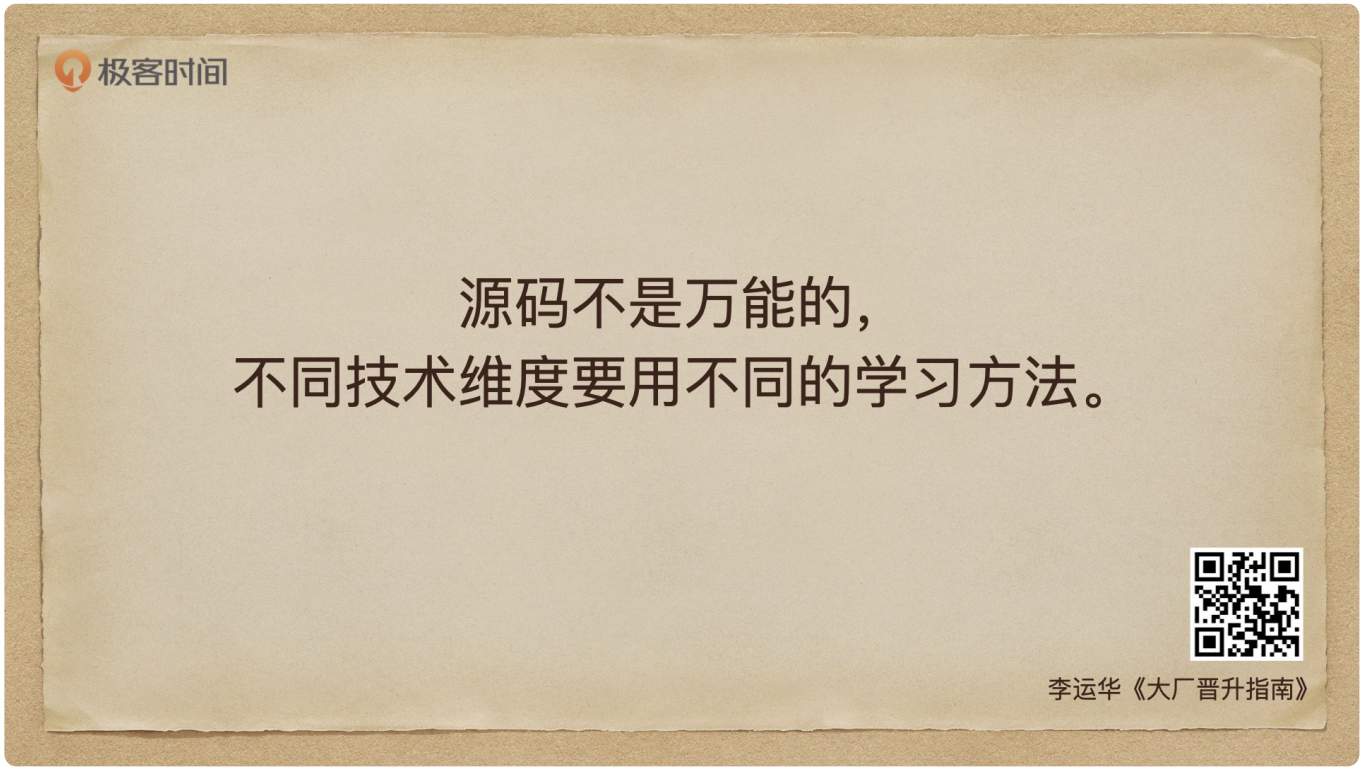
1. 环式学习法是构建闭环，打出组合拳，主要用来提升技术广度。
2. 环式学习法的步骤包括：先把闭环画出来，然后由近及远，逐步攻克闭环上的各个节点。
3. 环式学习法的优点有：培养全局视野，避免盲目地广撒网却捞不到鱼。



思考题

这就是今天的全部内容，留一道课后思考题给你吧。在你的面试或者晋升的过程中，有没有因为某个专业方面的问题没答上来而留下遗憾的经历？学完今天的内容后，你觉得可以用什么方式来学习，避免以后再留下类似的遗憾呢？

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。



提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 18 | 三段分解法：怎么利用10000小时成为大牛？

下一篇 20 | Play & Teach：怎么摆脱“从入门到忘记”的学习困境？

精选留言 (6)

写留言

天天立flag，月月打自...
2021-01-11

关于学习方法，最近看过一本书，书中有些学习方法别具一格，颠覆认知，书名——《认知天性：让学习变得轻而易举的心理学规律》。

这本书介绍了各种高效学习的方法，其中有几种方法，比较颠覆认知，书中认为学习时...
展开

作者回复: 我最近刚看完这本书，里面提到的学习方法我自己其实一直就是这样做的。我会3～4个主题同时进行学习，每个主题每次都学30分钟左右，一段时间后又回过头来复习加深一下。

例如我最近同时在学：ClickHouse、Redis专栏、推荐系统专栏、朝鲜战争、认知天性。

4 10

大土豆
2021-01-11

问netty的问题，最后的归宿一定是Linux内核，用户态和内核态的数据拷贝，IO多路复用，epoll的内核实现(红黑树挂载fd)，CPU时间片切换的原理，还有零拷贝。深入网络的话，TCP和UDP又能问的更多了，最后基本上可以把Linux内核给问完了。
展开

作者回复: Linux内核远远不止netty相关的内容哦。

不过你列举的基本都是Netty深度相关的，一般晋级面评的时候这里面都可能被问到。

2

Monday
2021-01-13

链式学习法的第一步，就是要明确一项技术的深度可以分为哪些层。
具体来说，就是画出“领域分层图”和“细节分层图”。一开始你可能会觉得画不出来，这恰恰说明你对深度的理解还不够，而尝试画图本身就是一个梳理结构、强化认知的过程。
...
展开

作者回复: 已经写了加餐，等编辑打磨后很快上线

1 2

Harvey
2021-01-13

你好，华仔：
问题：关于链式学习 。如何才能画出一个技能的领域分层，能否提供具体的指导原则和标准?
举例：对于一个不了解netty的人，肯定是画不出来这个图，可能只会只会想到第二层，或者想到的层次的技术不对，这些都会影响链式学习的效果。所以我的思考是能画...
展开

作者回复: 如何画领域分层图已经写了加餐，等编辑打磨后很快上线。

设计原理和设计方案的区别：设计方案是设计原理的落地实现。例如Reactor是原理，Redis的设计方案是单Reactor单进程、Memcached的设计方案是多Reactor多线程、Nginx是多Reactor多进程、我自己用过单Reactor多线程

1

蛙
2021-01-12

感觉后台开发要学好多知识，华仔有推荐的学习线路吗?

作者回复: 后台这个词范围太大了，还是要自己结合工作需求来制定自己的学习计划

Sisyphus235
2021-01-11

技术深度图的扫描请问有什么好的方式鸟瞰全局吗？谢谢

作者回复: 没看到这个问题，可以把问题讲详细些